XvFAT filesystem Development

Machida, Hiroyuki
Sony Corp

2005.06.14

machida – AT – sm.sony.co.jp

Introduction about the xvfat

- Picked up by linuxdevice.com and slashdot.org
  - There are some misunderstandings...
- Q&A
  - xvfat is a new fs?
    - No, developed for 2.4 kernel at about 2 years ago.
  - Why FAT?
    - FAT is widely used, by commercial products like Memory card, DSC, CamCorder, Mobile phone and Portable music player. CE vendor need to support it.
  - Current 2.6 kernel already achieved the goal of xvfat
    - I hope it !. However, I don't know what 2.6 archived exactly.
    - At least, we need tests for production level and some improvements
      - E.g. Repeat over 1000 times un-plugging media test while write operation with simultaneous thread access , white box testing for critical situation and so on.

Background and History - 1

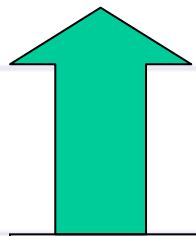
- Based on experience with Linux 2.2.2x or 2.4.8.
- Development was started early 2003 and finished at CY2003 3Q
- Targeting to 2.4.17. Later, ported to 2.4.20
  - Designed to work with a specific USB MemoryStick Adapter

Background and History - 2

- Address following issues; (met with 2.2.x/2.4.x)
  - Possible problems with removal media
    - Handling removed media during mount
      - Data could be corrupted, even it's file/dir are not in middle of operation
      - E.g. Entire FS could be corrupted if media removed while deleting files from two threads..
    - System behavior after media removed
      - Some system call works incorrectly based on cached data
      - No notification to user space
  - Char code inconsistency problem
    - 1-N mapping SJIS<-> unicode, in general
    - Directory cache may hit incorrectly
    - Possible problems on application side and interpretability
  - Other minor issues
    - timezone, dirtyflag

Improved points

- Minimize data corruption
  - Change order of changing data and meta-data
    - Assuming EraseBlock as an atomic operation unit
    - Cancellation of I/O elevator for block device to accomplish it
  - Change write system call behavior to wait write completion
- Return errors after media removed
- Invalidate cache correctly on media removal
- Notify event to user space
  - add new ioctl and /proc entry
- Improve Japanese file name support
- Use dirty flag for FAT32
- Add TIME ZONE support



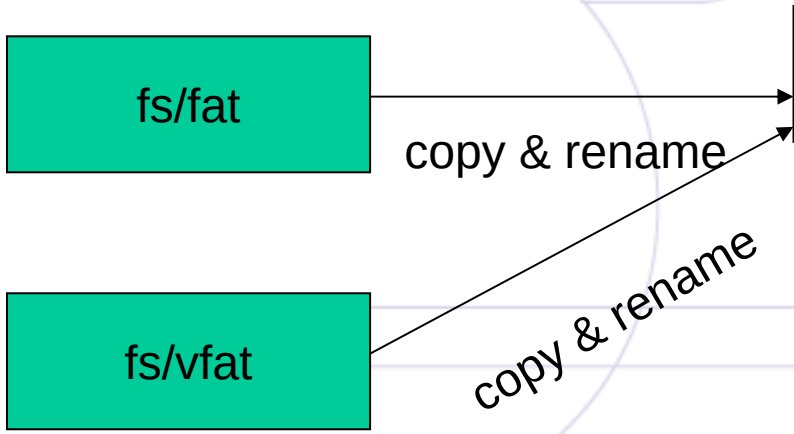
Current xvfat implementation

- Explanations for current xvfat implementation to open up discussions about new implementation on top of 2.6 kernel

- Design
  1. Co-existing with existing FAT implementation
  2. Suppose EraseBlock is unit of atomic operation
     - Update Fat is safe, if Fat resides in one EraseBlock
  3. Transaction control to fs operations.
     - Isolate Meta data operations from different threads.
  4. Use own implementations
     - Transaction control function
     - Elevator function to cancel reordering

1. source code organization

- FS designed to be coexisting with conventional FAT



modified:

```
include/linux/msdos_fs_i.h |   4
include/linux/msdos_fs_sb.h |  11
fs/fat/Makefile            |  12
fs/fat/buffer.c            |  39
fs/fat/cache.c             | 366 ++++++-
fs/fat/cvf.c               |  50 -
fs/fat/dir.c               | 228 +++-
fs/fat/fatfs_syms.c        |  41
fs/fat/file.c              | 170 +++
fs/fat/inode.c             | 562 +++++++++---
fs/fat/misc.c              | 129 +-
fs/vfat/namei.c            | 842 ++++++++++++++---
fs/vfat/vfatfs_syms.c      |  52 -
```

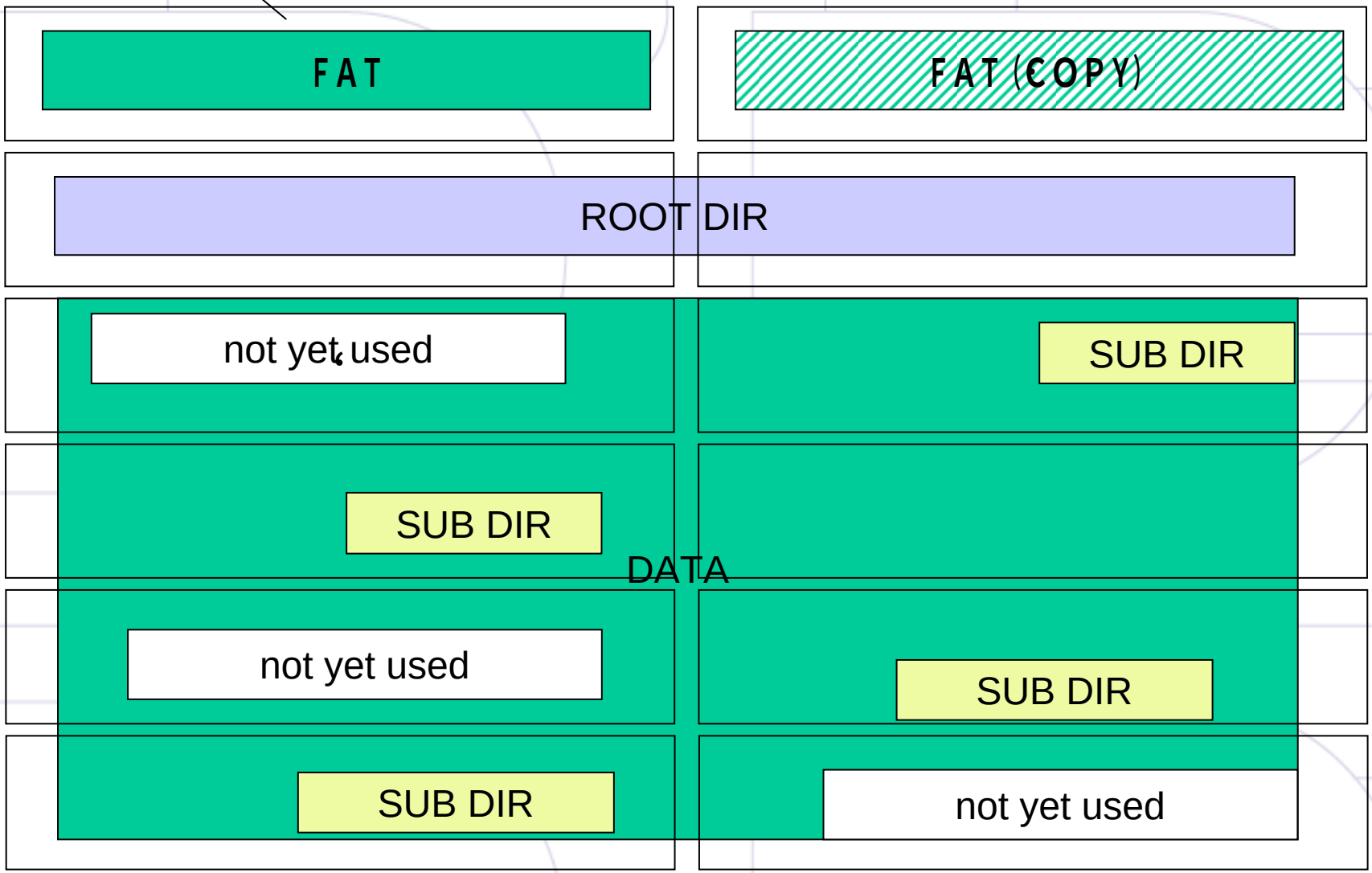
added:

```
2020 fs/xvfat/fwrq.c
1015 fs/xvfat/rdchk.c
 610 fs/xvfat/tz.c
  50 include/linux/rdchk.h
  41 include/linux/fat_tz.h
 224 include/linux/xvfat_fs.h
 245 include/linux/fwrq.h
```

- The elevator function modified too, under drivers/

```
elevator.c  |   36 ++++++++++++++++++++++++++++++++++
ll_rw_blk.c |   45 ++++++++++++++++++++++++++++++++++++++++++---
```

2. FAT FS organization & EraseBlock



3. Transaction control to fs operations

```
 struct buffer_head *xfat_extend_dir_ts(struct inode *inode,
*                                       fwrq_transaction_t *ts,
*                                       int allow_reorder)
 {
                :
*       ts2 = fwrq_transaction_get(inode->i_dev, FWRQ_TS_NOLOCK);
                :
*       nr = xfat_add_cluster_ts(inode,ts2);
                :
        sector = MSDOS_SB(sb)->data_start + (nr - 2) * cluster_size;
        last_sector = sector + cluster_size;
                :
        for ( ; sector < last_sector; sector++) {
*               bh = xfat_getblk(sb, sector);
                memset(bh->b_data, 0, sb->s_blocksize);
*               xfat_set_uptodate(sb, bh, 1);
*               fwrq_transaction_add_tail(ts, bh);
                        :
        }
*       fwrq_transaction_concat(ts, ts2);
                :
        inode->i_size += 1 << MSDOS_SB(sb)->cluster_bits;
        MSDOS_I(inode)->mmu_private += 1 << MSDOS_SB(sb)->cluster_bits;
*       fwrq_add_queue_inode(inode);
        return res;
 }
```

4-1. Transaction control function

- fwrq_add_queue - Add BH into tail of queue
- fwrq_flush_queue- flush BHs in queue
- fwrq_invalidate - invaildates and abandons all TRANSACTIONS and QUEUES belonging to DEV
- fwrq_transaction_get - get an new transaction data area.
  - This allocates and initialize new TRANSACTION, belongingto DEV. If TS_LOCK specified in flags, set LOCK bit in TS and lock DH.
- fwrq_transaction_add_tail - record an new operation as a part of transaction
  - Record an new opration specified by BH, into TRANSACTION.
- fwrq_transaction_concat - concatinate transactions
- fwrq_transaction_append1 - forced concatinate transactions (with reordering)
- fwrq_transaction_commit_to_queue - commit transaction to block device queue.
  - move recored block oerations as transaction into device block queue, then free transaction.
- fwrq_transaction_fix - commit transaction however never free it.
- fwrq_transaction_commit_to_queue_and_flush - commit transaction,

4-2. Elevator function to cancel reordering

```
+int elevator_inorder_merge(request_queue_t *q, struct request **req,
+                        struct list_head * head,
+                        struct buffer_head *bh, int rw,
+                        int max_sectors)
+{
+       struct list_head *entry;
+       unsigned int count = bh->b_size >> 9;
+
+       if (list_empty(&q->queue_head))
+               return ELEVATOR_NO_MERGE;
+
+       entry = &q->queue_head;
+       if ((entry = entry->prev) != head) {
+               struct request *__rq = blkdev_entry_to_request(entry);
+               if (__rq->cmd == rw &&
+                   __rq->rq_dev == bh->b_rdev &&
+                   __rq->nr_sectors + count <= max_sectors &&
+                   __rq->waiting == 0 &&
+                   __rq->sector + __rq->nr_sectors == bh->b_rsector) {
+                       *req = __rq;
+                       return ELEVATOR_BACK_MERGE;
+               }
+       }
+
+       *req = blkdev_entry_to_request(q->queue_head.prev);
+       return ELEVATOR_NO_MERGE;
+}
+
```

Moving to Kernel 2.6

- Working with the community
  - Isolate the patches by each feature
  - Utilize existing mechanism inside kernel w or w/o modification
- New features, if possible
  - Address generic removable media and devices
  - and more....
    - Support embedded storage, for example
- Now, open discussion...

Design and implementation issues - 1

- 0. Detection & Notification of unplugging
  - Issues
    - Media unplug - Poll media status or event
    - Device unplug - Recover and record I/O error needed
    - Notification event - Communication to user space
    - Status report - Keep track media/device status
  - Possible solution with 2.6
    - HOTPLUG can cover everything for unplug?
    - Time-out-handler in the ll_rw_block.c for device unplug?
    - LKML Patch [IOCHK interface for I/O error handling/detecting] for device unplug?
    - Notification using /proc, /sysfs or light weight methods?
- 1. Change source code organization
  - Utilize and share fs/fat and fs/vfat code

Design and implementation issues - 2

3. Suppose EraseBlock is unit of atomic operation
   - Not atomic in general.
   - Still use EraseBlock based operations ?
     - It can achieve good co-operations with underlying TransLayer?
     - It can utilize burst mode operations on NAND flash
   - Cluster allocation also could take account EraseBlock?
     - For example, prefer to select EraseBlock which has a series of empty sectors from beginning...
   - How to get EraseBlock size or size for optimal operational unit?
     - hardsect_size on blk dev request queue could be used?

Design and implementation issues - 3

4-1. Transaction control to fs operations.

- Possible implementation on 2.6
  - Use journaling-API (JBD)?
    - Doc could be produced by Documentation/DocBook/journal-api.tmpl
    - Use RAM area as external journal to record transactions
      - » RD can be used for this without deadlocking ?
    - Journaling FAT fs could be implemented by using internal journaling
  - JBD can be used、maybe, but the modification would be needed
    - Data submitting timing control may be needed with own implementation
    - JFS may uses own kernel daemon, not using kjournald
- Synchronous fs operations
  - to be a mount option, not default behavior

4-2. Elevator function to cancel reordering

- enable /disable reordering in a EraseBlock
- 2.6 kernel built-in evaluators are sufficient ?
- or Need porting in-order elevator to 2.6 ?

Other issues

- Need cooperation with under laying TransLayer, if possible
  - Reading by Write Block unit could achieve good error recovery, than giving up using all data in a Erase block.
  - Using data copies inside in TransLayer (eg cache) could achieve good error recovery.
- Adding journaling function to FAT sounds good
  - We have to take care that the log does not consist with data about the media while removed, written by PC.
  - Need to more investigate about advantages - Is there the difference between embedded case
    and hot plug case ?
- Need to more investigate about difference between HDD support,USB Mass Storage support,and embedded flash memory support
  - seek time, existence of TransLayer, EraseBlock length、WriteBlock length, and so on